ARCHORG: MODULE 5 & 6

M5T1: Cache Memory

Main Memory Characteristics

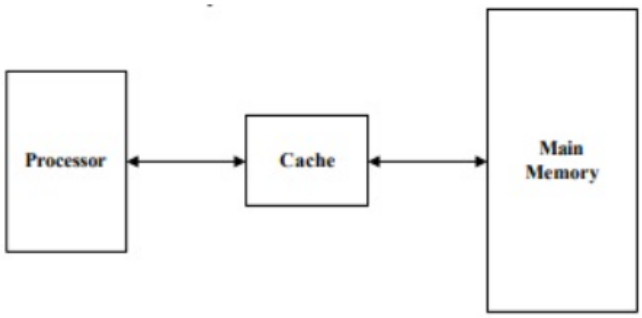
Bipolar Static RAM (SRAM) – Fast but expensive

Dynamic RAM (DRAM) – Cheaper but Slower

Solution: use DRAM for main memory and SRAM for cache memory.

Why use Cache Memory?

- Processor speed >> main memory speed
- Cache reduces access time
- Cache makes main memory appear faster to the processor.



Locality of Reference

The effectiveness of the cache mechanism is based on a property of computer programs called the ***locality of reference***.

Locality of reference manifest itself in two ways: ***temporal*** and ***spatial***

- **Temporal strategy**: keep recently used items in cache. Temporal means that a recently executed instruction is likely to be executed again very soon.
- **Spatial strategy**: Fetch blocks instead of single items**.** The spatial aspect means that instructions in close proximity to a recently executed instruction are also likely to be executed soon.

Operations of Cache Memory

Temporal vs Spatial

The **temporal aspect** of the locality of reference suggest that whenever an information item (instruction or data) is first needed, this item should be brought into the cache where it will hopefully remain until it is needed again.

The **spatial aspect** suggest that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that reside in adjacent addresses as well. The term block is used to refer to a set of contiguous address location of some size.

Replacement Algorithm

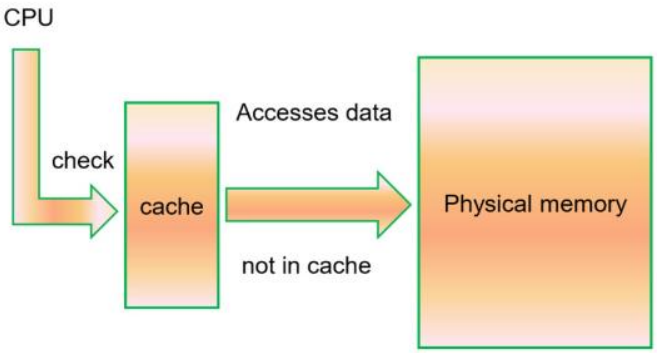
- When the processor issues a read request, the system transfers the contents of a block of memory words containing the location specified into the cache one word at a time.
- When the cache is full and the program references a memory word that is not in the cache, the cache control hardware must decide which block to remove to create space for the new block that contains the referenced word. The collection of rules for making this decision constitute the replacement algorithm.

Cache Miss and Cache Hit

- **Cache Hit**
    - If the requested word is in the cache, the Read or Write operation is performed on the appropriate cache location. In this case, a ***red hit*** or ***write hit*** is said to have occurred.
    - In a Read operation, the main memory is not involved. Only the cache is read.
    - In a Write operation, the system can proceed in two ways:
        1. **write - through protocol** - the cache location and the main memory location are updated simultaneously.
        2. **Write - back or copy-back protocol** - to update only the cache location to mark it as updated (using associated flag bit called the ***dirty*** or ***modified*** bit). The main memory location of the word is updated later, when the

block containing this marked word is to be removed from the cache to make room for new block.

- **Cache Hit** refers to data is found at a given memory level
- **Cache Miss**
    - When the addressed word in a Read operation is not in the cache, a ***read miss*** occurs. The block of words that contains the requested word is copied from the main memory into the cache. This approach, which is called ***load-through***, or ***early restart***.
    - During the Write operation, if the addressed word is not in the cache, a ***write miss*** occurs.



Cache Miss refers to data that is not found at a given memory level.

Types of Cache Misses

- ***Cold (also known as "Cold start or First reference) miss*** - it refers to a cache miss that occurs on first accesses to given blocks.
    - **Block** - It refers to the smallest transferable unit of information that are present in between memory hierarchies.
- ***Conflict (also known as "Collision or interference") miss*** - It refers to a cache miss that occurs when the cache is large enough, but multiple data objects are all map to the same set or block frame.
- ***Capacity miss*** – It refers to a cache miss that occurs when the set of active cache blocks (working set) is much larger than the cache capacity

M5T2: Cache Mapping Functions

Cache Mapping Functions – a mapping function specifies where to place a main memory block in the cache.

Formulas:

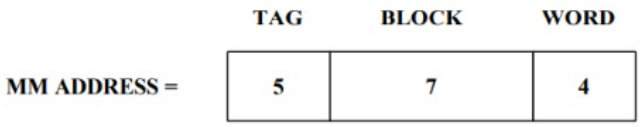
No. of Main Memory Blocks = MM Size / Block Size

No. of Cache Blocks = Cache Size / Block Size

Direct Mapping

In this technique, block k of the main memory maps onto block k modulo m of the cache where m is the number of cache blocks.

The memory address can be divided into three fields:



Formulas:

No. of Main Memory Blocks = MM Size / Block Size

No. of Cache Blocks = Cache Size / Block Size

No. MM Blocks per Cache Block = # of MM blocks / # of Cache blocks

No. of Tag Bits = $\log_2$(# of MM blocks per cache block)

No. of Block Bits = $\log_2$(No. of Cache Blocks)

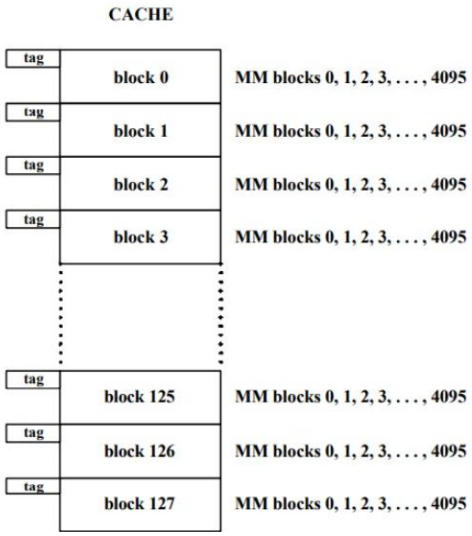
No. of Word Bits = $\log_2$(Block Size)

Associative Mapping

A main memory block can potentially reside in any cache block position.

The memory address can be divided into two fields: TAG and WORD.

ARCHORG: MODULE 5 & 6



No. of Main Memory Blocks = MM Size / Block Size

No. of Cache Blocks = Cache Size / Block Size

No. of Tag Bits = $\log_2$(No. of MM blocks)

No. of Word Bits = $\log_2$(Block Size)

M5T3: Cache Mapping Functions

Direct Mapping

- Only one fixed cache line
- Fast (use index to locate)
- Low (fixed position)

Associative Mapping

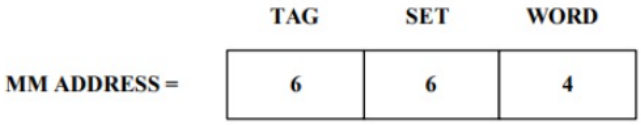
- Can go into any cache line
- Slower (checks all lines)
- High (No restrictions)

Set Associative Mapping

- A combination of Direct Mapping and Associative Mapping.
- Is a cache method where each block maps to a specific set

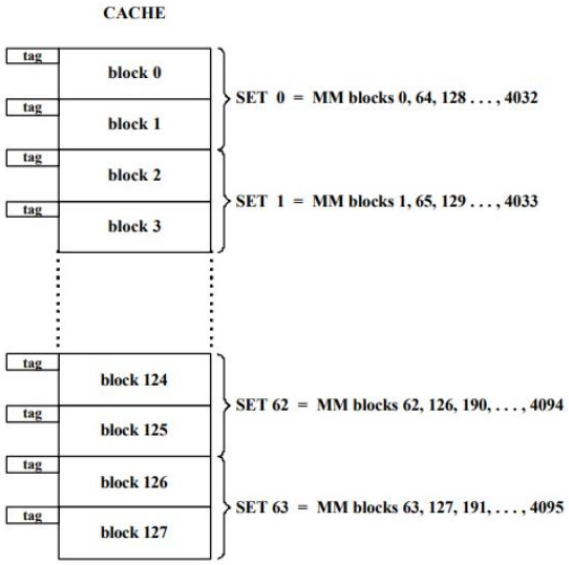
Blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any cache block of a specific set

The memory address can be divided into three fields:

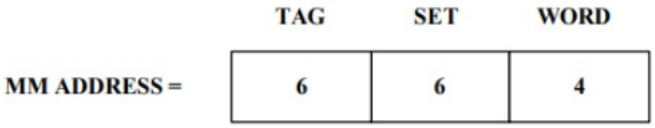


Blocks of cache are grouped into sets, and the mapping allows a block of main memory to resides in any cache block of a specific set.

Example: Assume 2 cache blocks per set



The memory address can be divided into three fields:



Formulas:

No. of Main Memory Blocks = MM Size / Block Size

No. of Cache Blocks = Cache Size / Block Size

No. of Sets = No. of Cache Blocks / No. of Blocks to a Set

No. of MM Blocks per Set = No. of MM Blocks / No. of Sets

No. of Tag Bits = $\log_2$(No. of MM Blocks per Set)

No. of Set Bits = $\log_2$(No. of Sets)

No. of Word Bits = $\log_2$(Block Size)

ARCHORG: MODULE 5 & 6

Block - Set Associative - Example

A 16 MB main memory is to be upgraded by adding a cache memory of 128 KB. Both the main memory and the cache memory are partitioned into blocks of 512 bytes. Assume that there are 8 blocks to a set. How many bits make up the tags, set and word fields of a memory address?

Solution:

No. of MM Blocks = $2^{24} / 2^9 = 2^{15}$ or 32, 768 blocks

No. of Cache Blocks = $2^{17} / 2^9 = 2^8$ or 256 blocks

No. of Sets = 256 / 8 = 32

No. of MM Blocks per Set = $2^{15} / 2^5 = 2^{10}$ or 1,024

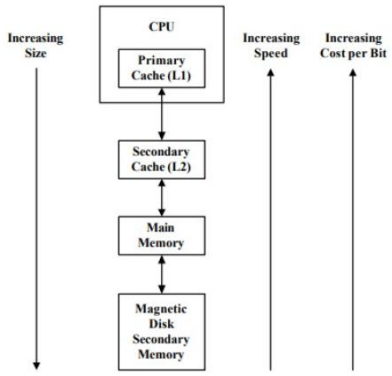
No. of Tag Bits = $\log_2(2^{10}) = 10$

No. of Set Bits = $\log_2(2^5) = 5$

No. of Word Bits = $\log_2(2^9) = 9$

MEMORY HIERARCHY

The entire memory can be viewed as the hierarchy depicted below:



The fastest access to data is through the processor registers. If registers are considered to be part of the memory hierarchy, then the processor registers are at the top in terms of speed of access. Of course, the registers provide only a minuscule portion of the required memory.

At the next level is the cache memory. There are two types of cache memory. A ***primary cache*** is located on the processor chip. This cache is small, because it competes for space on the CPU chip, which must implement many other functions. The primary cache is referred to as ***Level 1*** (***L1***) cache.

A ***secondary cache*** is placed between the primary cache and the main memory. It is referred to as ***Level 2*** (***L2***) cache.

Including a primary cache on the processor chip and using a larger, off-chip, secondary cache is the most common way of designing computers. However, other arrangements can be found in practice. It is possible not to have a cache on the processor chip at all. Also, it is possible to have both L1 and L2 caches on the processor chip.

The next level in the hierarchy is the main memory. It is much larger but significantly slower than the cache memory. In a typical memory, the access time for the main memory is about ten times longer than the access time for the L1 cache.

Disk devices provide a huge amount of inexpensive storage (secondary storage). They are very slow compared to the semiconductor devices used to implement main memory.

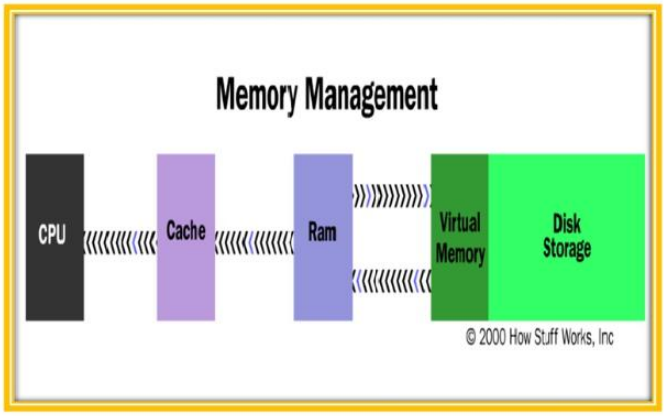
Performance Consideration

- Two key factors in the commercial success of a computer are **performance and cost.**
- A common measure of success is the ***price/performance ratio.***
- Performance depends on how fast machine instructions can be brought into the processor for execution and how fast they can be executed.

ARCHORG: MODULE 5 & 6

M6T1: Cache and Virtual Memory Replacement Policies

Cache Memory versus Virtual Memory



Difference of Cache Memory and Virtual Memory

Cache Memory

- It is defined as a **small, fast temporary storage area** of a recently accessed instruction and data that acts as a **buffer** between the CPU and main memory.
- It works by **moving data** to and from physical memory.
- **Primary goal: Increase SPEED**

Virtual Memory

- Also known as "**Imaginary Memory Location"**
- It is defined as an area on the second storage device (also known as "**page file**"), that holds chunks of the program that are brought into main memory (RAM as needed).
- It works by **swapping data** in and out of physical memory.
- **Primary goal: Increase SPACE**

Mapping – the mechanism by which **virtual addresses are converted into real memory addresses**.

Virtual address – this refers to an address used by the **programmer** and **produced by the CPU**.

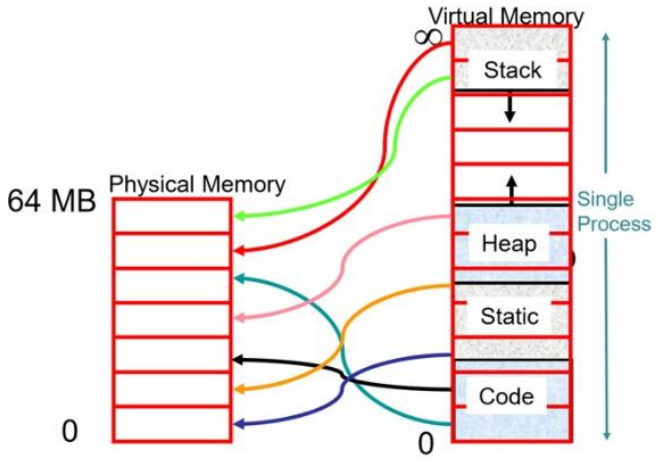
Real (Memory or Physical) address – This refers to an address of word in physical memory.

Virtual Memory Implementation

Techniques in Implementing Virtual Memory

1. Paging

- It is a process of **copying a virtual page** from a disk to an equal-sized **frame** in main memory.
  - **Page** – it is a fixed-length block which *contains the **virtual address** that is transferred as a unit between RAM and Disk Storage*.
  - **Frames** – it is an ***area of RAM that holds a page*.**



2. Segmentation

- It is a process of dividing a virtual **address space** into variable – length segments, often under the control of the programmer and can be used as a convenience for organizing the programs and data (i.e code segment and data segments).
  - **Address space** – it is a range of address of the memory and I/O that can be referenced.
- **Combination of both (Pagin ang Segmentation)**

Replacement Policies

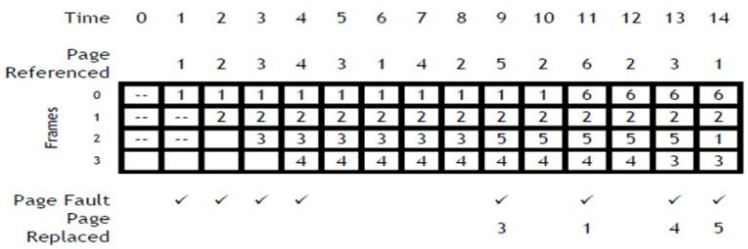
These are used to determine which if the several existing blocks (cache memory) or page (virtual memory) are can be overwritten if there are no more available slots in which to place a block or page.

Cache Replacement Options

1. Least Recently Used (LRU)

ARCHORG: MODULE 5 & 6

- It uses an **aging method** which determines the time the block were last accessed and **replaced those block that has not been access and has been unused for the longest period of time**.
- It has to **maintain an access history** for each block, which ultimately **slows down the cache**.



2. Random

- It picks a block at random and replaces it with a new block

Writing and Updating a Cache

Two problems to contend with when replacing a certain block in a cache

- More than one device is accessing the main memory
- Multiple CPU's are attached to the same bus and each CPU has its own local cache

Write Through

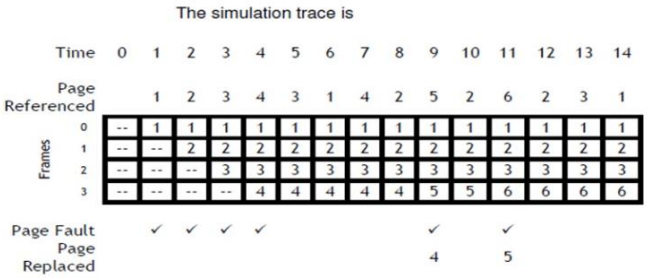
- It is strategy that ensures that a **data is always written to memory and the cache** at the **same time**.
- It would not improve performance for writes over a system without a cache because it incurs the cost of a memory access for each write.
- Disadvantages: **Creates bottleneck**.

Write Back

- It is a strategy where data is written to the **cache block only**, and the modified cache block is **written to memory only when it is replaced in the cache**.

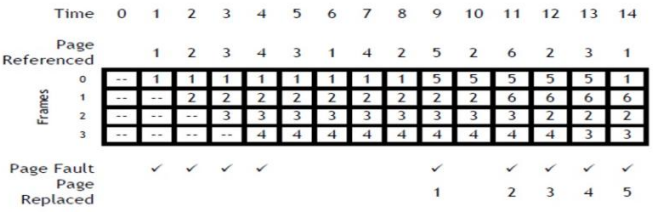
Optimal

- It replaces the page which **will not be used for the longest** (**future**) period of time.
- **Produces the lowest possible page fault rate**
- Impossible to implement since it requires future knowledge of reference string.
  - **Reference String –** It is a sequence of the pages on the disk that are demanded to swap into the memory



First In First Out (FIFO)

- It replaces the block that was added to the set first.



ARCHORG: MODULE 5 & 6

M6T2: Multiprocessors and Multiple Computers

Multiprocessor

- Multiple processors that share memory
- Connected via some interconnection network plus the software which contains program (known as the ("**parallel processing program**") that runs on it simultaneously.
- Operates together on a single problem.

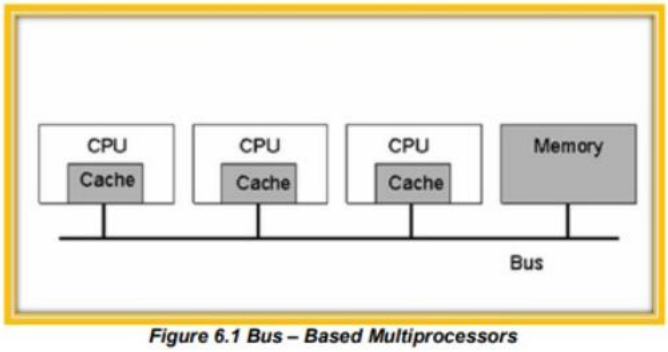


Figure 6.1 Bus – Based Multiprocessors

Figure 6.1 shows a bus - based multiprocessor composed of several CPUs with high speed cache memory and memory connected to a common bus

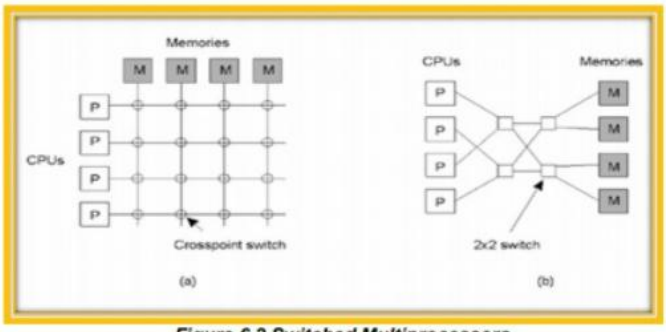


Figure 6.2 Switched Multiprocessors

Figure 6.2 shows a switched multiprocessor. Figure (a) shows a crossbar switch which connects the CPUs and different memory modules. Figure (b) shows an omega network (a network that contains 2x2 switches which has two inputs and two outputs) which connect the CPU and memories.

Multiple Computer (Multicomputers)

- Multiple standalone single or dual processor commodity computers that don't share memory
- Connected by High-Speed Communication Network
- Share a critically demanding computational task.

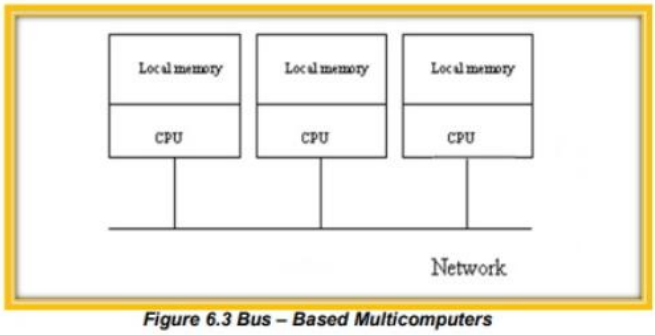


Figure 6.3 Bus – Based Multicomputers

Figure 6.3 shows a bus - based multicomputers composed of independent computers in which their own CPUs have its own local memory.

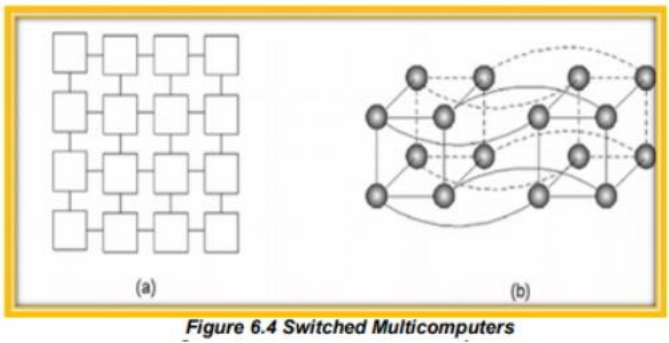


Figure 6.4 Switched Multicomputers

Figure 6.4 shows switched multicomputers. Figure (a) shows a grid that connects several computers. Figure (b) shows a four - dimensional hypercube that connects several computers in which their own CPU has n connections to other CPUs.

Ways to Use Multiprocessors

1. **Flynn's Classification Scheme**

- It is one of the more widely used parallel computer classification, since 1966
- It distinguishes multiprocessor computers according to the dimension of:
  - **Instruction stream** – It is defined as a flow of instruction from memory to the CPU.
  - **Data stream** – It is defined as a flow of operands between the memory and the processing unit bi-directionally.

2. **Single Instruction and Single Data (SISD) stream**

- A conventional, Serial (Non-Parallel) Computers.
- Single instruction: Only one instruction stream is acted on by CPU during any one clock cycle.
- Single Data: Only one data stream is used as input during any one cloak cycle

ARCHORG: MODULE 5 & 6

- Examples:
    - CDC 6600
    - CDC 7600
    - Amdhal 470/6
    - Cray-1

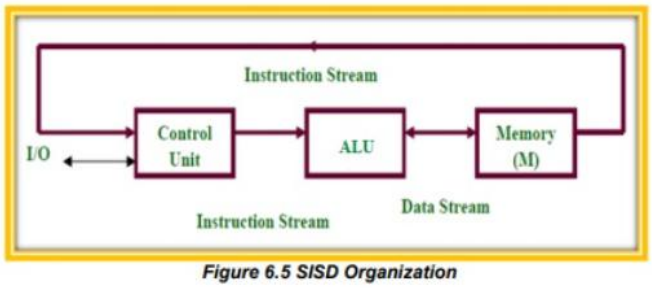


Figure 6.5 SISD Organization

Figure 6.5 shows a SISD Organization wherein a sequential executions is performed by one CPU containing a single processing element and ALU under one control unit.

3. **Single Instruction and Multiple Data (SIMD) Streams**

- **Single Instruction**: All processor units execute the same instruction at any given clock cycle
- **Multiple data**: Each processing unit can operate on a different data element
- Examples:
    - ILLIAC-IV DAP
    - PEPE Connection Machine (CM-1)
    - BSP
    - STARAN
    - MPP

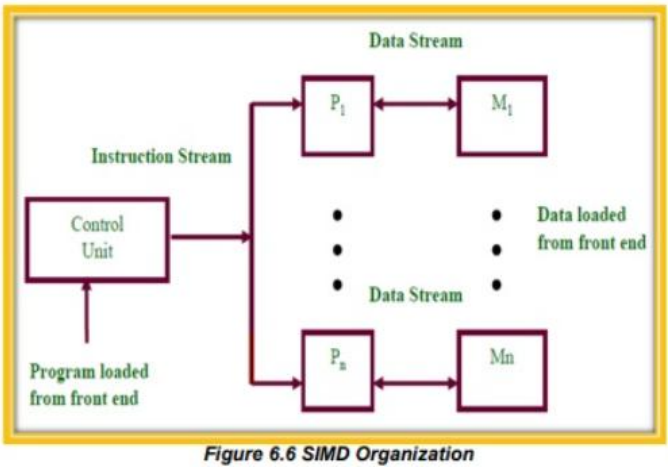


Figure 6.6 SIMD Organization

Figure 6.6 shows an SIMD Organization in which multiple processing elements receive the same instruction broadcast from the single control unit.

4. **Multiple Instructions and Single Data (SIMD) Organization**

- **Multiple instruction**: Each processing unit operates on the data independently via independent instruction streams
- **Single data**: A single data stream is fed into multiple processing units
- Examples:
    - C.mmp built by Carnegie-Mellon University

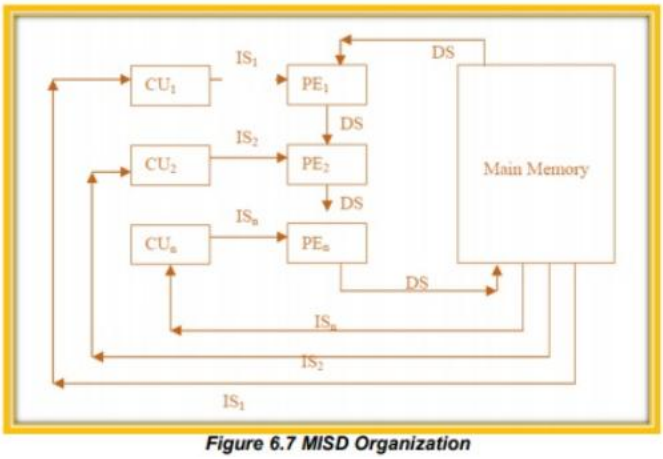


Figure 6.7 MISD Organization

Figure 6.7 shows a MISD Organization in which multiple processing elements are organized under the control of multiple control units.

5. **Multiple Instructions and Multiple Data (MIMD) Streams**

- **Multiple instruction:** Every processor may execute a different instruction stream
- **Multiple Data**: Every processor may work with a different data stream
- Examples:
    - C.mmp IBM 370/168 MP
    - Burroughs D825 Univac 1100/80
    - Cray-2 Tandem/16
    - S1 IBM3081/3084
    - Cray X-MP C.m*
    - HEP BBN Butterfly
    - Pluribus FPS T/40000
    - iPSC

ARCHORG: MODULE 5 & 6

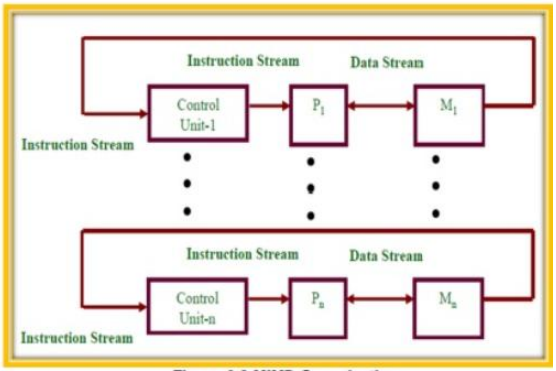


Figure 6.8 MIMD Organization

Figure 6.8 shows a MIMD Organization in which multiple processing elements and multiple control units are organized as in MISD. The difference however is that, multiple instruction streams in the organization operates on multiple data streams. The processors work on their own data with their own instructions.

Centralized vs. Distributed Shared Memories

- **Centralized Shared Memory**
    - It is also known as the "uniform memory access" (UMA) or "symmetric (shared-memory) multiprocessor"(SMP)
    - It has a symmetric relationship to all processors
    - It has a uniform memory access time from any processor

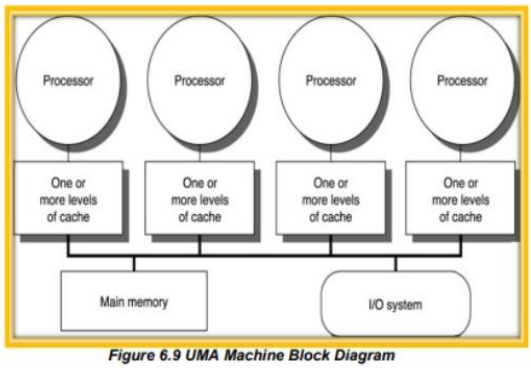


Figure 6.9 UMA Machine Block Diagram

Figure 6.9 shows a uniform memory access architecture in which the processor share memory through the interconnection network. Before each processor can access a certain memory element, the data from that processor is temporarily stored in the cache. There is a dedicated cache memory for each processor.

- **Distributed Shared Memory**
    - It is also known as the "non-uniform memory access" (NUMA)
    - It does not allow uniform access to all shared memory locations but instead, each processor can access the memory module closest to it

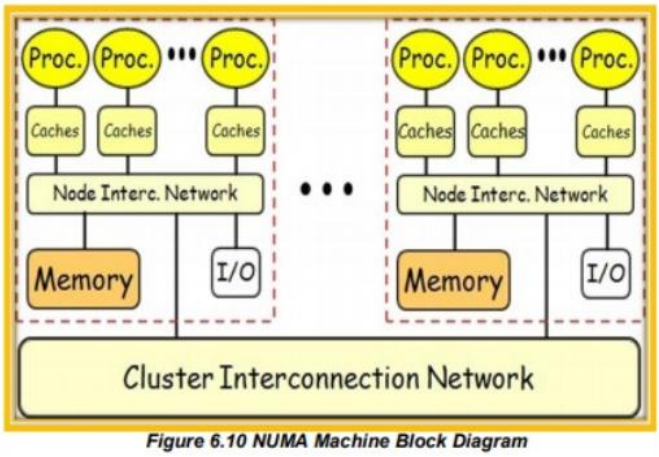


Figure 6.10 NUMA Machine Block Diagram

Figure 6.10 shows a NUMA Machine Block Diagram in which processor can directly access its own memory but can still access memories from one processors closest to it. The access to its own memory is much faster because it is a direct access method while accessing memories from other processors is a lot slower due to the latency of connection from the interconnection network and the processor handling the memory.

Cache Coherence

- It refers to the state of the system in which all caches have up to date copies of the memory blocks

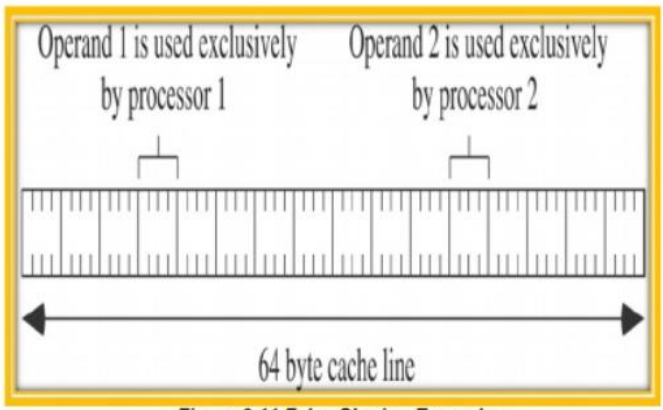


Figure 6.11 False Sharing Examples

ARCHORG: MODULE 5 & 6

Figure 6.11 shows example of a false sharing which happens when two operands that are not shared between processes share the same cache line. Each process will invalidate the other's cache line when writing data without a real need, unless the compiler prevents this.

Solutions to Cache Coherence

- **Cache Enforcement Strategy –** It is a mechanism which makes caches consistent

Hardware Solutions Categories

- **Snoopy Cache Protocols –** It relies on a broadcasts consistency commands via bus which results to high communication contention and limits scalability
- **Directory Based Cache Protocols –** It does not rely on a bus to exchange coherence information which results to communication contention, reduction and are more scalable. In designing a directory definition of the following are required:
    - **Cache Block Granularity** – It is the size of the cache and the size of a cache line
    - **Cache Controller Design –** It refers to a hardware that maintains the directory and processes memory request
    - **Directory Structure** – It refers to how a cache and memory information is organized
- **Lock Based Protocols –** Also know as cache – coherent network architecture. Its coherence information exchanged by reading and writing notices from the lock which protects the shared memory. This results to more scalability than directory protocols

Hardware Solutions Policies

- **Write – invalid –** In this policy, only first write invalidates; subsequent writes to do require invalidate signals.

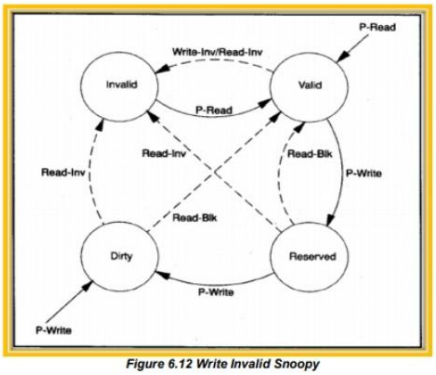


Figure 6.12 Write Invalid Snoopy

Figure 6.12 shows that state diagram of a write invalid snoopy. The program read (P-Read) determines weather the read is valid or invalid. If valid, the P-Read goes to the program write(Write - Inv/Read-Inv) and invalidates other read and write states. If not valid, the P-Read continues on doing the program read until it becomes valid. Once the program write (P-write) is valid with a reserved memory, it repeats the process of program write until it is allowed to write the data in the memory.

- **Write – update –** In this policy, the writing processor broadcasts the new data over the bus and all copies are then updated. This reduces cache latency seen at processors.

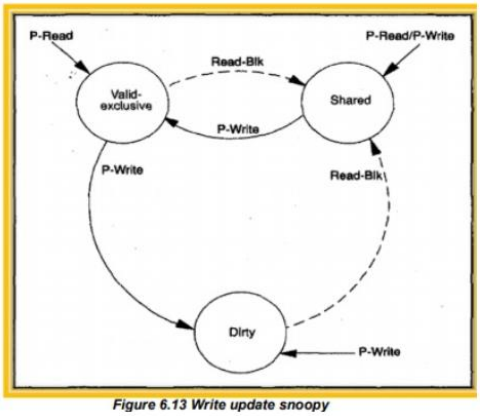


Figure 6.13 Write update snoopy

Figure 6.13 shows the state diagram of a write update snoopy. A program read (P-Read) looks for an exclusive and valid process and redirects it to program write (P-write) to be written on the dirty block. The read block (Read - Blk) then goes to the shared block wherein two P-reads can access the block. The process permitted will receive the exclusive and valid block and becomes ready for P-write.

ARCHORG: MODULE 5 & 6

Consistency commands

- **Consistency Model –** It defines how the consistency of data values is maintained.
- **Software Protocols –** It enforces consistency with required hardware mechanism by relying either on the compiler or specialized software handlers